Введение

Прикладное программирование

кафедра
прикладной и компьютерной оптики

Список литературы

- **Дейтел, Дейтел. Как программировать на C++: Пятое издание. М.: Издательство «Бином-Пресс», 2008. -1456с.**
- Страуструп Б. Язык программирования C++ / Б. Страуструп. – СПб.: "Невский диалект", М.: Бином, 2008. – 1104 с.
- Мейерс С. Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов - М.: ДМК Пресс, 2000. -240с.
- Мейерс С. Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов - М.: ДМК Пресс, 2000. -304с.
- Мейерс С. Эффективное использование STL. Библиотека программиста. - Спб.: Питер, 2002. - 224 с.

- книги на диске example
- электронный учебник на сайте кафедры

Основы языка C++

Прикладное программирование

кафедра
прикладной и компьютерной оптики

Пример программы, выводящей текст на экран (пример 1)

```
////////////////////////////////////////////////////////////////////////////
// подключение библиотеки ввода-вывода
#include <iostream>
// подключение стандартного пространства имен для использования библиотек
using namespace std;

////////////////////////////////////////////////////////////////////////////
// исполнение программы всегда начинается с функции main
void main()
{
    // вывод текста на экран
    cout<<"Welcome to C++!"<<endl;
    cout<<"Welcome"<<endl<<"to"<<endl<<"C++!"<<endl;
}
////////////////////////////////////////////////////////////////////////////
```

Пример программы, выводящей текст на экран (пример 1)

- **Директивы препроцессору**
  - Подробное описание других функций стандартной библиотеки приводится в главе 2.
- **Ввод и вывод на экран**
  - Подробное описание стандартной библиотеки ввода-вывода приводится в разделе 2.2.
- **Комментарии**

```
/* многострочный
комментарий */

c++; // однострочный комментарий до конца строки
```

- **Функции**
  - Подробно работа с функциями рассматривается в разделе 1.9.
  - Полный список ключевых слов языка C++ приведен в приложении 3.

Пример программы сложения целых чисел (пример 2)

```
////////////////////////////////////////////////////////////////////////////
// подключение библиотеки ввода-вывода
#include <iostream>
// подключение стандартного пространства имен для использования библиотек
using namespace std;

////////////////////////////////////////////////////////////////////////////
// функция main начинает исполнение программы
void main()
{
    // объявления переменных
    int number1, number2;
    int sum=0;

    cout<<"Enter first integer: "; // запросить данные
    cin>>number1; // прочитать первое число в number1

    cout<<"Enter second integer: "; // запросить данные
    cin>>number2; // прочитать второе число в number2

    sum = number1 + number2; // сложить числа; записать сумму в sum

    cout<<"Sum is "<<sum<<endl; // вывести сумму
}
////////////////////////////////////////////////////////////////////////////
```

Переменные и их объявление

- **Переменная** – это место в памяти компьютера, где может сохраняться некоторое значение для использования его в программе.
  - полный список основных типов данных приведен в приложении 2

- **Идентификатор** – это последовательность символов, состоящая из латинских букв, цифр и символов подчеркивания (_), обозначающая имена переменных

```
i   // обычно целая переменная цикла
count  // количество
COUNT  // другая переменная
buff_size  // составное (из 2 слов) имя с символом _
g374  // непонятно
_foo  // плохо
if  // не может быть идентификатором, т.к. это ключевое слово
374q  // не может начинаться с цифры
```

Объявление и инициализация переменных

- Объявление переменной - определение ее типа
- Инициализация переменной - присваивание ей первоначального значения

```
int i; // объявление и определение типа (declaration)
i=1;   // инициализация (initialization)

int j=1; // объявление, определение и инициализация
int i=0,j; // объявление нескольких переменных



// В операторах вывода можно производить вычисления
cout<<"Sum is "<<number1 + number2<<endl;
```

Арифметические операторы

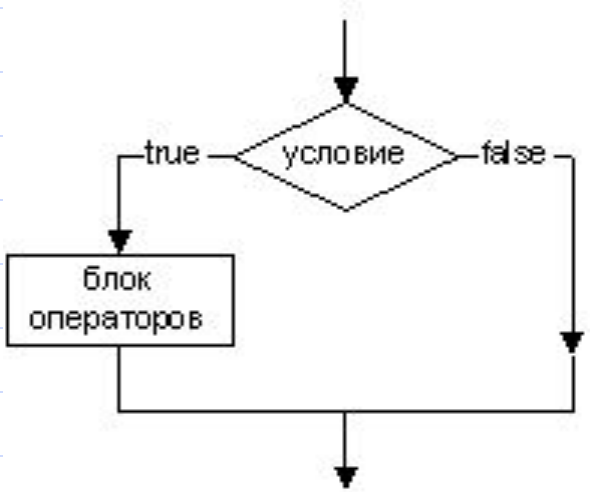
- Примеры арифметических операций (**пример 3**)
- Арифметические операторы
  - **бинарные** - в выражении участвуют два операнда, например x=a+b
  - **унарные** - в выражении участвует один операнд, например x=-a
- Круглые скобки используются для группировки выражений таким же образом, как в алгебраических выражениях

```
a*(b+c)
```

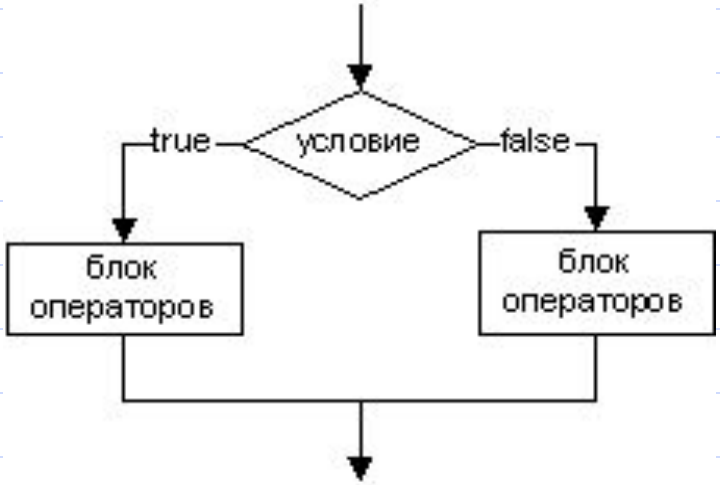
- Полный список арифметических операций и таблица их старшинства приведена в приложении 1.

Условные конструкции

```
if(условие)
{
  блок операторов;
}
```




```
if(условие)
{
  блок операторов 1;
}
else
{
  блок операторов 2;
}
```



Пример условных конструкций

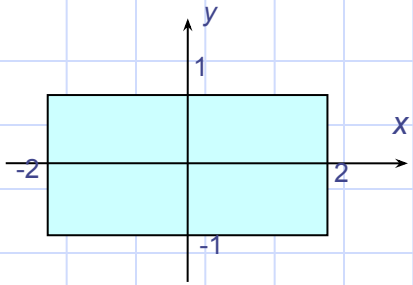
```
////////////////////////////////////////////////////////////////////////////////
// функция main начинает исполнение программы
void main()
{
    // объявление переменных
    int x,y;

    cout<<"Enter two integers to compare: ";  // запросить ввод
    cin>>x>>y;  // прочитать два введенных числа

    if(x == y)   // число x равно числу y ?
        cout<<x<<" == "<<y<<endl;
    if(x != y)   // число x не равно числу y ?
        cout<<x<<" != "<<y<<endl;
    if(x < y)    // число x меньше числа y ?
        cout<<x<<" < "<<y<<endl;
    if(x > y)    // число x больше числа y ?
        cout<<x<<" > "<<y<<endl;
    if(x <= y)   // число x меньше либо равно числу y ?
        cout<<x<<" <= "<<y<<endl;
    if(x >= y)   // число x больше либо равно числу y ?
        cout<<x<<" >= "<<y<<endl;
}
////////////////////////////////////////////////////////////////////////////////
```

Логические выражения

- Логическое выражение может принимать два значения:
  - true (истинно)
  - false (ложно)
- Пример попадания точки с координатами x, y в прямоугольник (**пример 5**):



  - Логические операции:

```
&& - логическое И
|| - логическое ИЛИ
! - логическое НЕ
```

Логические операции И, ИЛИ, НЕ

- Математическая запись условия, что координата x лежит внутри прямоугольника:
  - $-1 < x < 1$

- Запись на C++:

```
x<2 && x>-2
```

  - или

```
(x<2) && (x>-2)
```

- x НЕ внутри прямоугольника

```
(x >2) || (x<-2)
```

  - или

```
!( (x <2)&&(x >-2) )
```

выражение 1	выражение 2	(выражение 1) **&&** (выражение 2)	(выражение 1) **||** (выражение 2)
true	**true**	**true**	**true**
true	false	false	**true**
false	**true**	false	**true**
false	false	false	false

Типичные ошибки в написании условий

```
////////////////////////////////////////////////
if(x=1) // неправильно! выполняется всегда!
{
    y=x+3;
    z=y*5;
}
////////////////////////////////////////////////
if(x==1); // неправильно! выполняется всегда!
{
     y=x+3;
     z=y*5;
}
////////////////////////////////////////////////
if(x==1) // неправильно!
     y=x+3;
     z=y*5;
```

```
if(x==1)
{
    y=x+3;
    z=y*5;
}
```

```
if(x==1)
{
  [пустой оператор];
}
y=x+3;
z=y*5;
```

```
if(x==1)
{
    y=x+3;
}
z=y*5;
```

Вложенные условия

- Пример вложенного условия

```
if ( x > 5 )
{
    if ( y > 5 )
        cout<<"x and y are > 5";
}
else
    cout<<"x is <= 5";
```

- Неправильное написание:

```
if( x > 5 )
   if( y > 5 )
      cout<<"x and y are > 5";
else
   cout<<"x is <= 5";
```

```
if ( x > 5 )
{
  if ( y > 5 )
  {
    cout<<"x and y are > 5";
  }
  else
  {
    cout<<"x is <= 5";
  }
```

Арифметический логический оператор

```
переменная = условие ? значение1 : значение2;
```

- Пример:

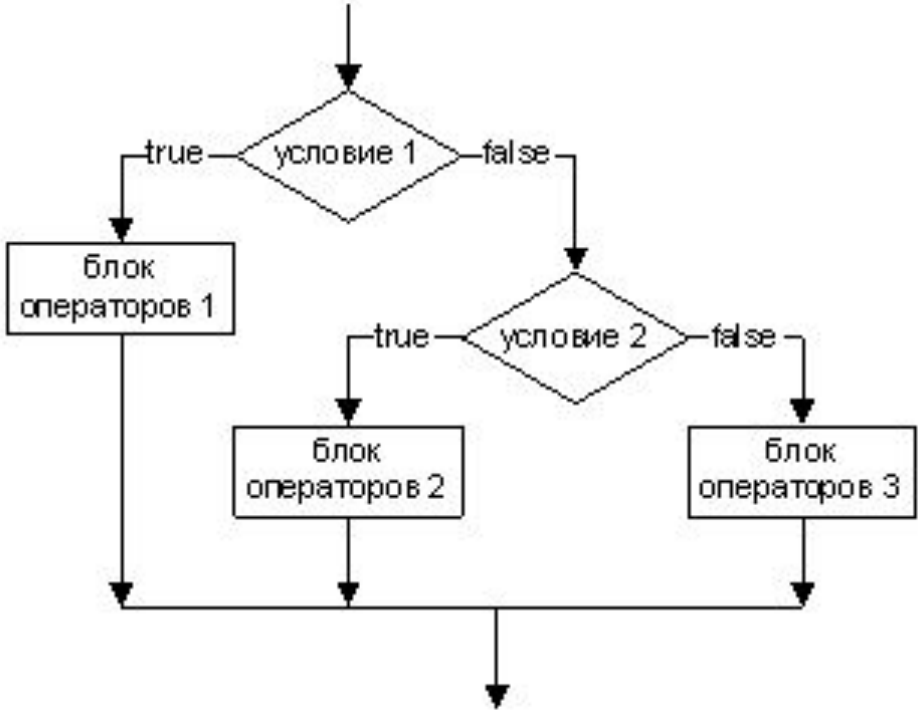
```
int i=3;
int j=(i>0) ? 1 : -1; // j=1
```

- То же, что:

```
if(i>0)
{
    j=1;
}
else
{
    j=-1;
}
```

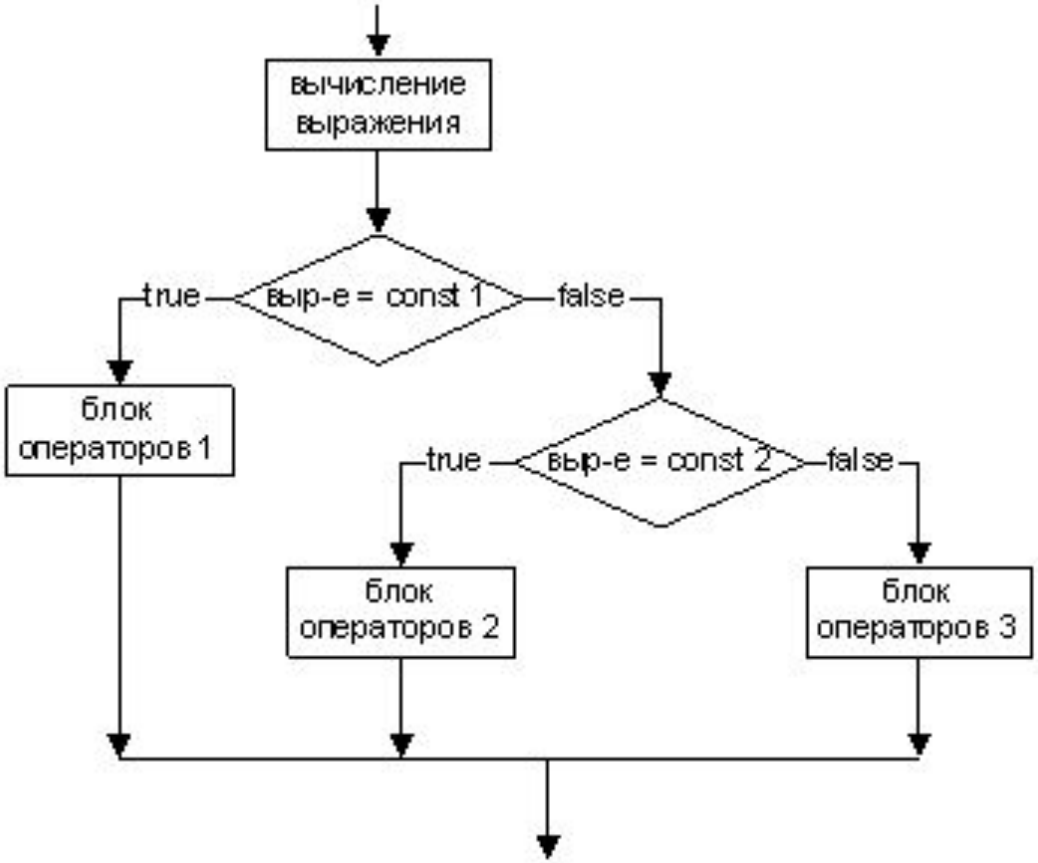
Селективные конструкции (if)

```
if(условие)
{
    блок операторов;
}
else if(условие)
{
    блок операторов;
}
else
{
    блок операторов;
}
```



Селективные конструкции (switch)

```
switch(переменная)
{
    case целая константа 1:
    {
        блок операторов;
        break;
    }
    case целая константа 2:
    {
        блок операторов;
        break;
    }
    default:
    {
        блок операторов;
    }
}
```



Примеры

- Пример определения оценки в зависимости от количества баллов (**пример 6**)
  - условия проверяются последовательно
  - если ни одно из условий не выполнено, выполняется блок else
    - если пропустить последний блок else – возможна ситуация когда ни одно из условий не выполнится.

- Пример меню с выбором действия (**пример 7**)
  - Каждое действие выполняется в зависимости от значения некоторого целого значения, которое может принимать переменная или выражение, проверяемое в операторе switch

Селективные конструкции (сравнение if и case)

```
switch(menu_number)
{
  case 1:
    {
       cout<<"a+b="<<a+b<<endl;
       break;
    }
  case 2:
    {
       cout<<"a-b="<<a-b<<endl;
       break;
    }
  case 3:
    {
       cout<<"a*b="<<a*b<<endl;
       break;
    }
  case 4:
    {
       cout<<"a/b="<<a/b<<endl;
       break;
    }
  default:
    {
       cout<<"Wrong menu item"<<endl;
    }
}
```

```
if(menu_number==1)
{
   cout<<"a+b="<<a+b<<endl;
}

else if(menu_number==2)
{
   cout<<"a-b="<<a-b<<endl;
}

else if(menu_number==3)
{
   cout<<"a*b="<<a*b<<endl;
}

else if(menu_number==4)
{
   cout<<"a/b="<<a/b<<endl;
}

else
{
   cout<<"Wrong menu item"<<endl;
}
```

Селективные конструкции (сравнение if и case 2)

```
switch(menu_number)
{
  case 1:
    {
        cout<<"a+b="<<a+b<<endl;
    }
  case 2:
    {
        cout<<"a-b="<<a-b<<endl;
        break;
    }
  case 3:
    {
        cout<<"a*b="<<a*b<<endl;
        break;
    }
  case 4:
    {
        cout<<"a/b="<<a/b<<endl;
        break;
    }
  default:
    {
        cout<<"Wrong menu item"<<endl;
    }
}
```

```
if(menu_number==1)
{
   cout<<"a+b="<<a+b<<endl;
}
else if(menu_number==1|| menu_number==2)
{
   cout<<"a-b="<<a-b<<endl;
}

else if(menu_number==3)
{
   cout<<"a*b="<<a*b<<endl;
}

else if(menu_number==4)
{
   cout<<"a/b="<<a/b<<endl;
}

else
{
   cout<<"Wrong menu item"<<endl;
}
```

Циклы while и do … while

- Оператор цикла позволяет программисту определить действие, которое должно повторяться, пока некоторое условие остается истинным
- Пример возведения в степень в цикле (**пример 8**)
  - значения product: 4, 8, 16, 32, 64, 128
  - Если не предусмотреть в теле оператора while действия, которое делает условие в while ложным, получается бесконечный цикл, в котором повторение никогда не заканчивается
- Перепишем пример 8 с использованием цикла do...while

Цикл с постусловием do...while

```
////////////////////////////////////////////////////////////////////////
  /
// функция main начинает исполнение программы
void main()
{
    // описание переменных
    int product = 2;
    // оператор цикла с предусловием
    do
    {
       product = product * 2;
       cout<<"product="<<product<<endl;
    }while(product <= 100);
}
////////////////////////////////////////////////////////////////////////
  /
```

- Результат такой же, как у цикла с предусловием - 4, 8, 16, 32, 64, 128
- Чтобы увидеть разницу между циклом с предусловием и постусловием, изменим примеры следующим образом:

Циклические конструкции (сравнение while и do … while)

```
// описание переменных
int product = 128;
// оператор цикла с предусловием
while(product <= 100)
{
    product = product * 2;
    cout<<"product="<<
            product<<endl;
}
cout<<"result product="<<
        product<<endl;
```

- результат:

```
result product=128
```

```
// описание переменных
int product = 128;
// оператор цикла с постусловием
do
{
    product = product * 2;
    cout<<"product="<<
            product<<endl;
} while(product <= 100);

cout<<"result product="<<
        product<<endl;
```

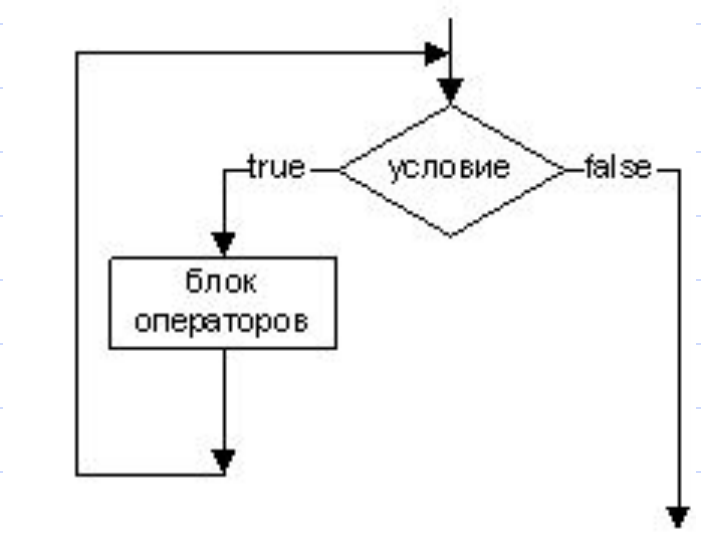
- результат:

```
256
result product=256
```

Циклические конструкции

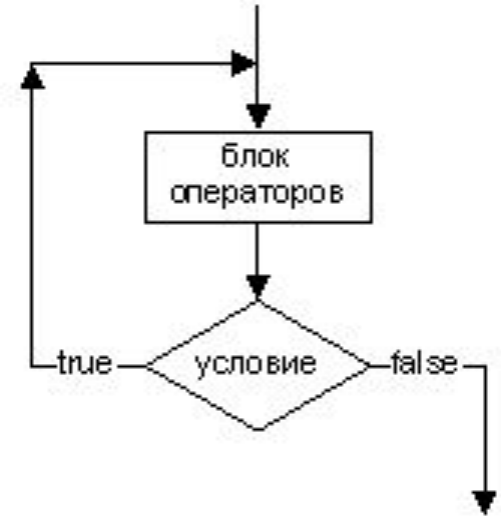
- Цикл с предусловием

```
while(условие)
{
    блок операторов;
}
```



- Цикл с постусловием

```
do
{
    блок операторов;
}while(условие);
```



Пошаговый цикл for

- Пошаговый цикл for позволяет выполнять блок операторов, заключенный в фигурные скобки задуманное количество раз.

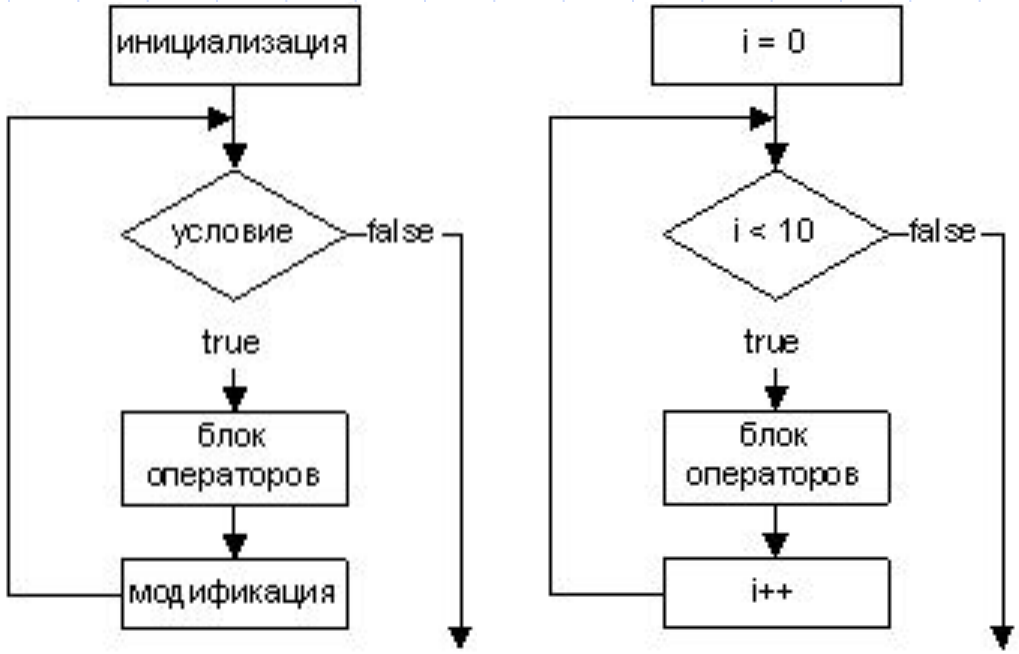
- Пример работы оператора for - вычисление суммы чисел (**пример 9**)

Пошаговый цикл

- Пошаговый цикл
  - **Инициализация** задает начальное значение счетчику цикла,
  - **Условие** определяет до каких пор цикл продолжает выполняться
  - **Приращение** модифицирует счетчик цикла

```
for(инициализация; условие; модификация)
{
    блок операторов;
}

for(int i=0; i<10; i++)
{
    x*=i;
}
```



Пошаговый цикл

- Переменная-счетчик должна иметь целочисленный тип данных
- Если переменная была объявлена в цикле, то по завершении цикла эта переменная будет уничтожена
  - **Область действия переменной** определяет, где в программе она может использоваться
- оператор for можно представить эквивалентным оператором while:

```
while ( условие )
{
    блок операторов;
    приращение;
}
```

Примеры пошагового цикла

- Еще несколько примеров заголовков цикла:
  - изменение управляющей переменной от 1 до 100 с шагом 1:

```
for(int i = 1; i <= 100; i++)
```

  - изменение управляющей переменной от 100 до 1 с шагом -1 (уменьшение на 1):

```
for(int i = 100; i > 0; i--)
```

  - изменение управляющей переменной от 7 до 77 с шагом 7:

```
for(int i = 7; i <= 77; i += 7)
```

  - изменение управляющей переменной от 20 до 2 с шагом -2:

```
for(int i = 20; i >= 2; i -= 2)
```

  - изменение управляющей переменной в последовательности: 2, 5, 8, 11, 14:

```
for(int j = 2; j <= 20; j += 3)
```

  - изменение управляющей переменной в последовательности: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0:

```
for(int j = 99; j >= 0; j -= 11)
```

Выражения инициализации и приращения и условия

- Выражения инициализации и приращения могут быть списками выражений, разделенных запятыми:

```
for(int x=0, y=0; x<10; x++, y++)
```

- Все три выражения в операторе for являются необязательными, например:

```
for(int i=0; i < 100; i++)
```

- можно записать:

```
int i=0;
for( ;  ; )
{
    if(i>=100)
        break;
    i++;
}
```

Операторы break и continue

- Когда оператор **break** исполняется в операторе while, for, do... while или switch, происходит немедленный выход из цикла (или switch)
- Исполнение оператора **continue** в операторе while, for или do... while вызывает пропуск оставшейся части тела оператора и переход к следующей итерации цикла
- Пример вычисление факториала (**пример 10**)

Использование функций библиотеки STL

- Использование функций библиотеки STL **(<u>пример 11</u>)**
  - функция pow() возводит число **3.14** в квадрат и присваивает полученный результат переменной power, где pow - имя функции; числа 3.14 и 2 — аргументы функции. В качестве аргументов функции может быть число или переменная соответствующего типа.
  - Аргументы перечисляются в скобках после имени функции, если аргументов несколько, то они отделяются друг от друга запятыми.

Определение функции

- Определение (реализация) функции

```
возвращ.тип имя (список аргументов с типами)
{
    инструкции
    return возвращ.знач.;
}
```

  - например:

```
double plus(double x, double y)
{
    return x+y;
}
```

- Тип возвращаемого значения может быть целым, вещественным, и т.д., кроме того, функция может не возвращать никакого значения - тип void.

```
void print_value(double res)
{
    cout<<"result is: "<<res<<endl;
}
```

  - Оператор return является обязательным только для функции, возвращающей значение

Объявление функции

- Объявление функции (прототип функции)

```
возвращ.тип имя (список аргументов с типами);
```

  - например:

```
double plus(double x, double y);
```

  - Прототипы обычно помещаются в отдельный заголовочный (header) файл или в начале файла
  - Компилятор сверяется с прототипом функции, для проверки соответствия:
    - количества аргументов
    - типы аргументов
    - тип возвращаемого значения

    - Каждый аргумент должен быть совместим с типом соответствующего параметра. Например, параметр типа double может принимать значения 7.35, 22 или -0.03456, но не строку (например "hello")

Обращение к функции

- **Обращение к функции (вызов функции)**

```
имя (список аргументов);
```

  - например:

```
double a, b, c;
a=plus(b, c);
```

    - Каждому параметру в определении функции (**формальный параметр функции**) должен соответствовать один аргумент в вызове функции

- Пример функции (**пример 12**)
  - функция сложения двух чисел
  - функция печати числа на экран
  - Значение функции plus можно присвоить какой-то переменной, или использовать прямо при вызове другой функции или при использовании оператора cout

Автоматическое размещение данных

- Память зарезервирована в exe-модуле
  - Определение Объявление = инициализация;

```
int i=1; //тип переменная=значение
```

- Время жизни
  - **глобальное** – в течение работы программы с переменной ассоциирована область памяти и обратиться к ней можно из любой точки программы
  - **статическое** – в течении работы программы с переменной ассоциирована область памяти, но обратиться к ней можно только из определенных точек программы
  - **локальное** – при каждом входе в блок { } для хранения переменной выделяется область памяти при выходе освобождается и теряет свое значение

```
// b не существует
if(a>0)
{
    int b=1;
}
// b не существует
```

Классы памяти

- extern - внешний (глобальный)
  - переменная глобальна для всех функций и доступна в любой точке программы
- static – статический
  - сохраняет предыдущее значение при повторном входе в блок { }
- auto – автоматический (по умолчанию)
  - при входе в блок { } память распределяется, а при выходе из блока память освобождается, значения теряются

- Переменные явно не инициализированные программистом
  - **extern** и **static** устанавливаются системой в нуль.
  - **auto** не инициализируется и может содержать "мусор".

- **const** - переменная инициализируется один раз после объявления и ее значение не модифицируемое

```
const double pi=3.141593;
```

См. пример программы

Ссылки

- **Ссылка** (reference) – это переменная особого вида, которая представляет собой альтернативное имя переменной (псевдоним)
  - ссылку нельзя объявить без инициализации

```
int i=1;
int& r=i; // r - новое альтернативное имя переменной i

int x=r;  // x=1
r=2;      // i=2

r++; // чему равно значение переменных r и i ?
```

См. пример программы

Указатели

```
int i=1; //тип переменная=значение
```

- Указатель (pointer) - это переменная особого вида предназначенная для хранения адреса объекта
  - Диапазон значений: положительные целые числа или null (0)
  - занимает 32 бита (4 байта)

- Объявление указателя

```
int *p;
```

  - означает, что переменная p имеет тип int * (т.е. указатель на int) и указывает на объект типа int

```
double *xPtr, *yPtr;
```

адрес ячейки	байт памяти
.....	.....
0x00A0	00000000
0x00A1	00000000
0x00A2	00000000
0x00A3	00000001
0x00A4	.....
.....	.....

Указатели

- Присваивание указателя

```
int i=1; // объявление целой переменной i
int *p;  // объявление переменной типа указатель,
         // т.е. значение по адресу p является целым числом
p=&i;    // p=0x00A0 (оператор получения адреса)
```

- Операция разименования (*)
  - получение значения переменной хранящейся по указанному адресу

```
int i=1;     // объявление целой переменной i
int *p ;     // объявление переменной типа указатель,
             // т.е. значение по адресу p является целым числом
p=&i;        // p=0x00A0 (оператор получения адреса)
int t=*p+1;  // эквивалентно t=i+1;
p++;         // прибавляем 1 к указателю - переходим к
             // следующей ячейке памяти с другой переменной
```

* - значение переменной по адресу

& - адрес переменной

См. пример программы

Передача параметров в функцию по ссылке и указателю

- **Передача по значению** (call by value)
  - в вызываемой функции создается копия аргумента
  - изменения копии внутри функции не влияют на значение исходной переменной вызывающей функции
- **Передача по указателю** (call by pointer)
  - передаются указатели на переменные, в которых хранятся значения
  - в вызывающей функции значения тоже изменяются
- **Передача по ссылке** (call by reference)
  - передаются ссылки (псевдонимы) на переменные
  - в вызывающей функции значения тоже изменяются

См. пример программы

Стандартная библиотека C++

Прикладное программирование

кафедра
прикладной и компьютерной оптики

Состав стандартной библиотеки C++

- Ввод/вывод
  - см. электронный учебник, раздел 2.2
- Математические функции (комплексные числа, случайные числа)
  - см. электронный учебник, раздел 2.4
- Строковый тип данных и форматные преобразования
  - см. электронный учебник, раздел 2.3
- Работа со временем и датой
  - Контейнеры, итераторы, обобщенные алгоритмы